CS598PS - Machine Learning for Signal Processing

Decision Theory and Simple Classifiers

23 September 2015

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Today’s lecture

- Decision theory
- Linear and quadratic classifiers
  - Gaussian models, the perceptron

Classification

- In detection we had one template
  - Decision: was it there?
    - Or rather, how much of it is there?

- In classification we have many templates
  - Decision: Which one is the most dominant?

Process overview

- We provide examples of classes
  - Training data
- We make models of each class
  - Training process
- We assign all new input data to a class
  - Classification

Making an assignment decision

- Face example
  - Dot products relate to a likelihood of match
  - This is linked to a probability

- Having a class probability for each face, how do we make a decision?

Motivating example

- Which do we pick?

$\mathbf{x}$ $\mathbf{y}$ $\mathbf{x}^\top \mathbf{y}$

Template face 1

Template face 2

Unknown face

0.94

0.87

Motivating example

- Template 1 is more "likely"

$\mathbf{x}$

Template face 1

Template face 2

$\mathbf{y}$

Unknown face

$$P\left(\mathbf{y} \mid \text{face}_1, \text{face}_2\right)$$

$\propto 0.94$

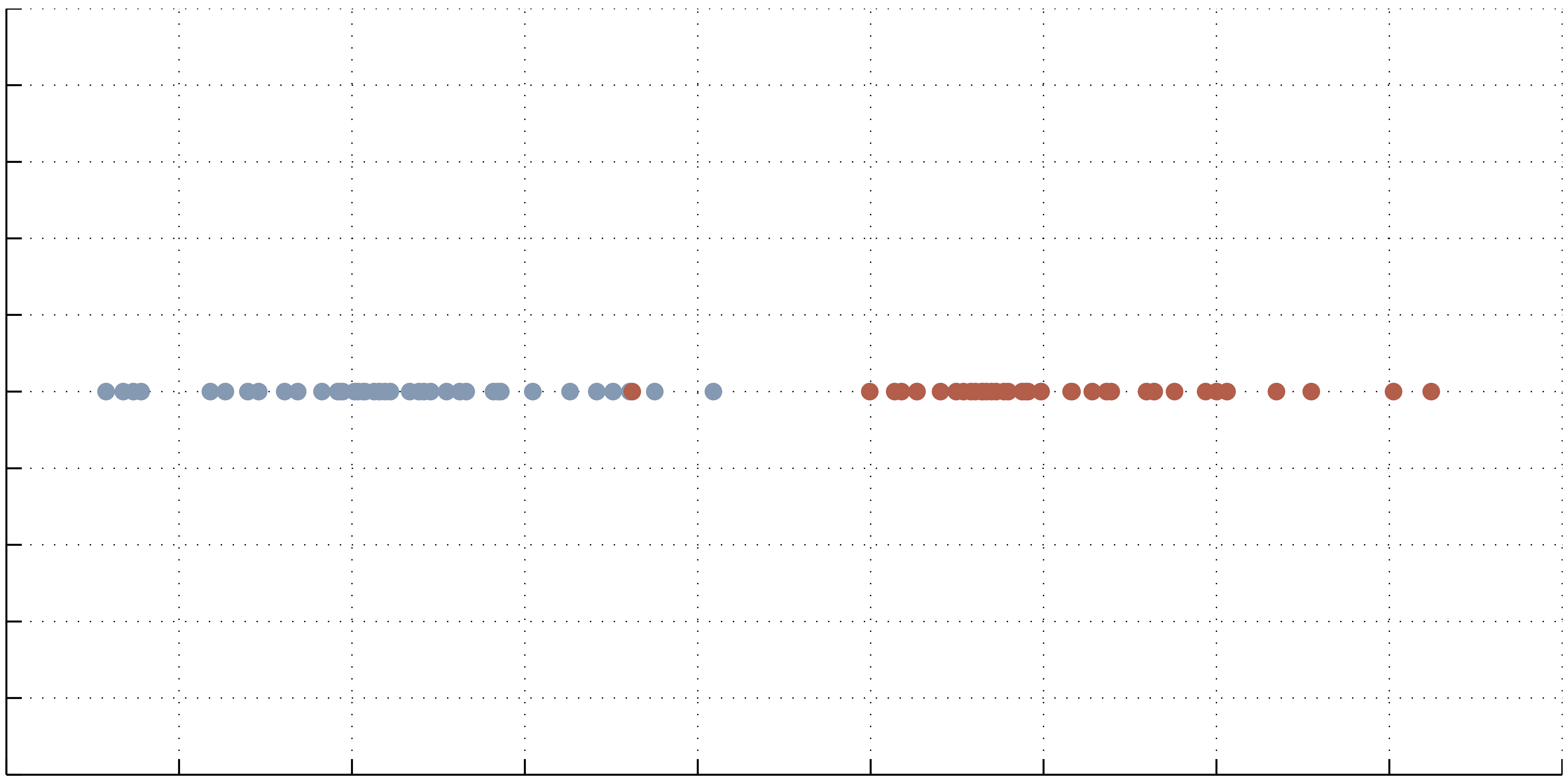
$\propto 0.87$

How the decision is made

- In simple cases the answer is intuitive
- To get a complete picture we need to probe a bit deeper
- Bayesian decision theory

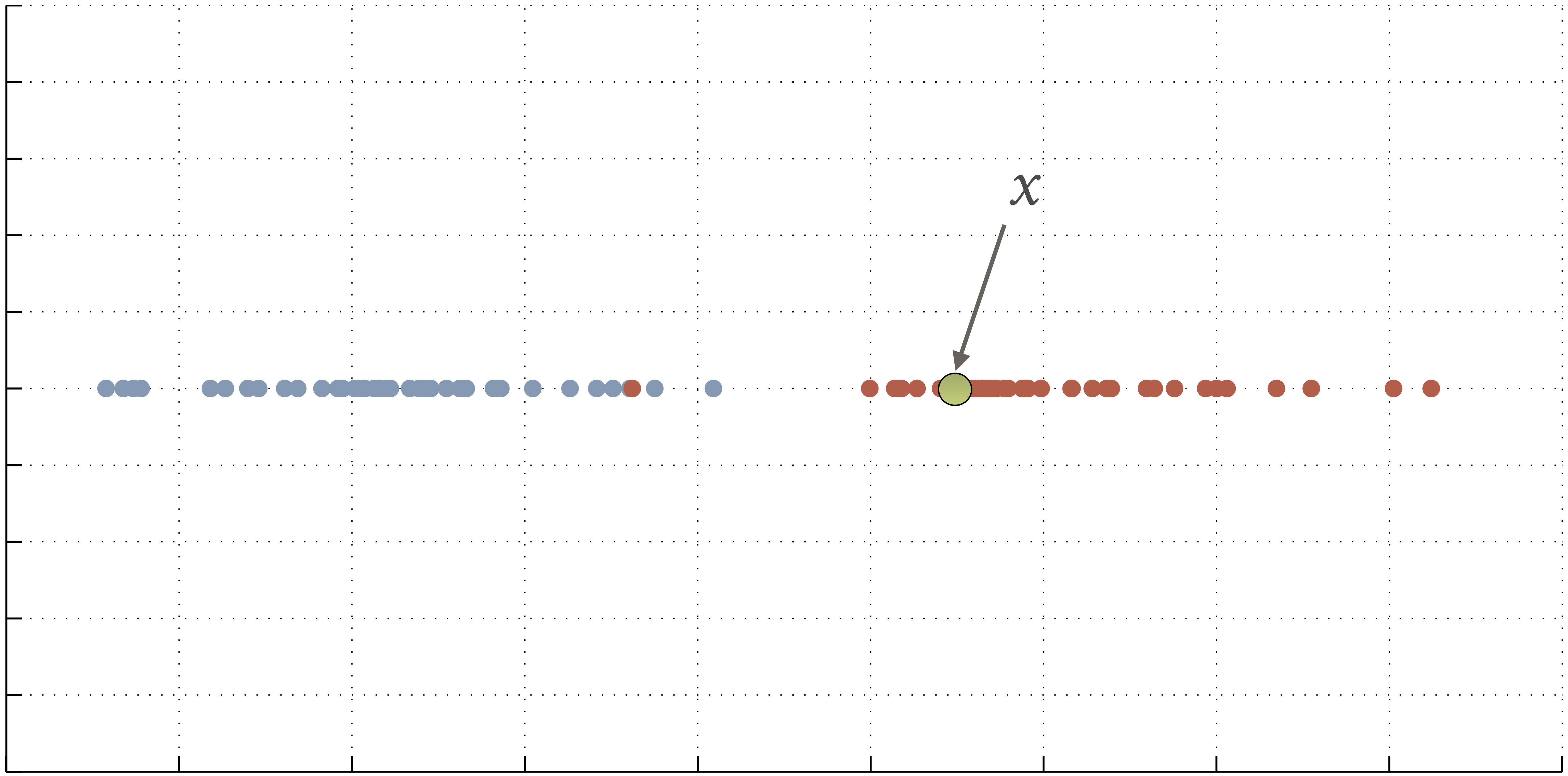
Starting simple

- Two-class case, ω_1 and ω_2
  - 1-dimensional data



Starting simple

- Given a sample x, is it ω_1 or ω_2?
  - i.e. $P(\omega_i \mid x) = ?$



Getting the answer

- The *class posterior probability* is:

$$P(\omega_i \mid x) = \frac{\overbrace{P(x \mid \omega_i)}^{\text{Likelihood}}\overbrace{P(\omega_i)}^{\text{Priors}}}{\underbrace{P(x)}_{\text{Evidence}}}$$

- To find the answer we need to fill in the terms in the right-hand-side

Filling the unknowns

- Class priors
  - How much of each class?

$$P(\omega_1) \approx N_1 / N$$
$$P(\omega_2) \approx N_2 / N$$

- Class likelihood: $P(x \mid \omega_i)$
  - Requires that we have a model of each ω_i
    - E.g. ω_i can be a Gaussian distributed so that:

$$P(x \mid \omega_i) = \mathcal{N}\left(x \mid \mu_{\omega_i}, \Sigma_{\omega_i}\right)$$

Filling the unknowns

- Evidence:

$$P(x) = P(x \mid \omega_1)P(\omega_1) + P(x \mid \omega_2)P(\omega_2)$$

- We now can estimate the class posteriors:

$$P(\omega_1 \mid x), P(\omega_2 \mid x)$$

Making the decision

- Bayes classification rule:

$$\text{If } P(\omega_1 \mid x) > P(\omega_2 \mid x) \text{ then } x \text{ belongs to class } \omega_1$$

$$\text{If } P(\omega_1 \mid x) < P(\omega_2 \mid x) \text{ then } x \text{ belongs to class } \omega_2$$

- Easier version:

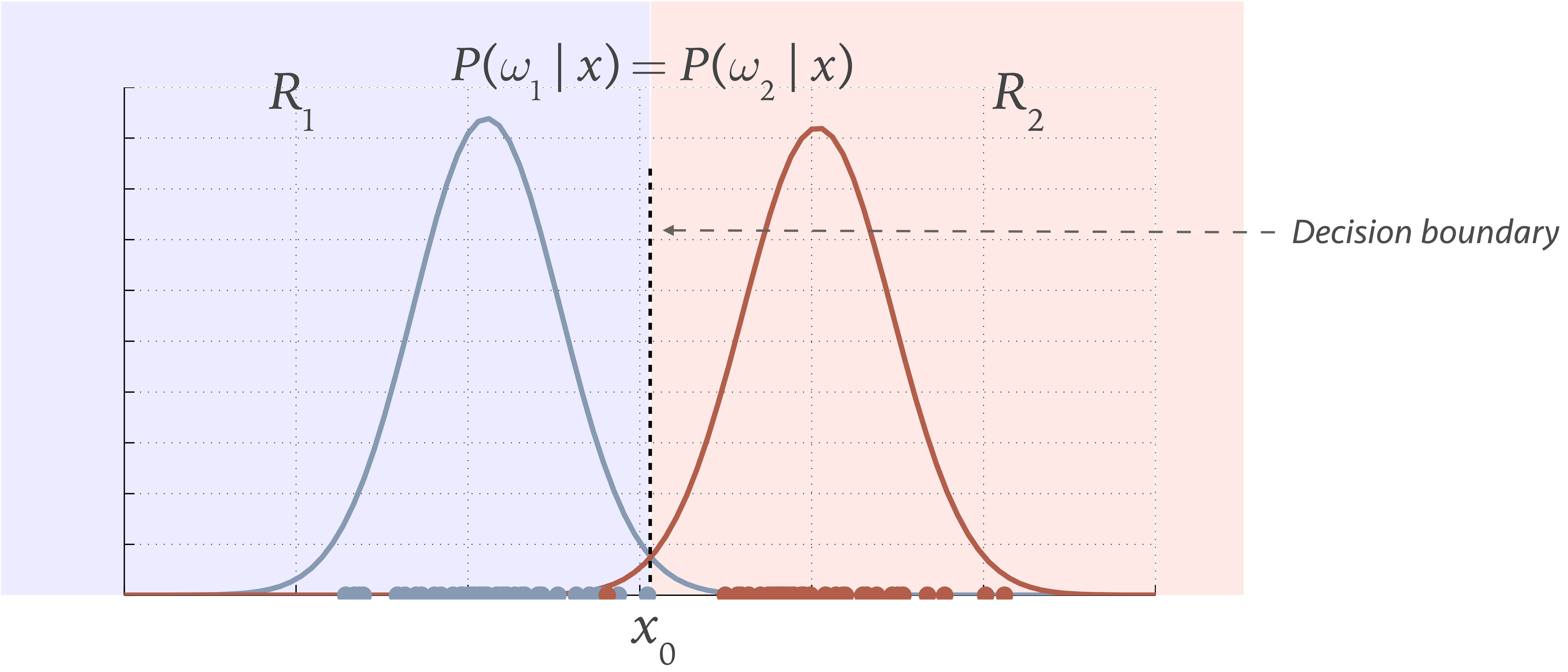
$$P(x \mid \omega_1)P(\omega_1) \gtrless P(x \mid \omega_2)P(\omega_2)$$

- Equiprobable class version:

$$P(x \mid \omega_1) \gtrless P(x \mid \omega_2)$$

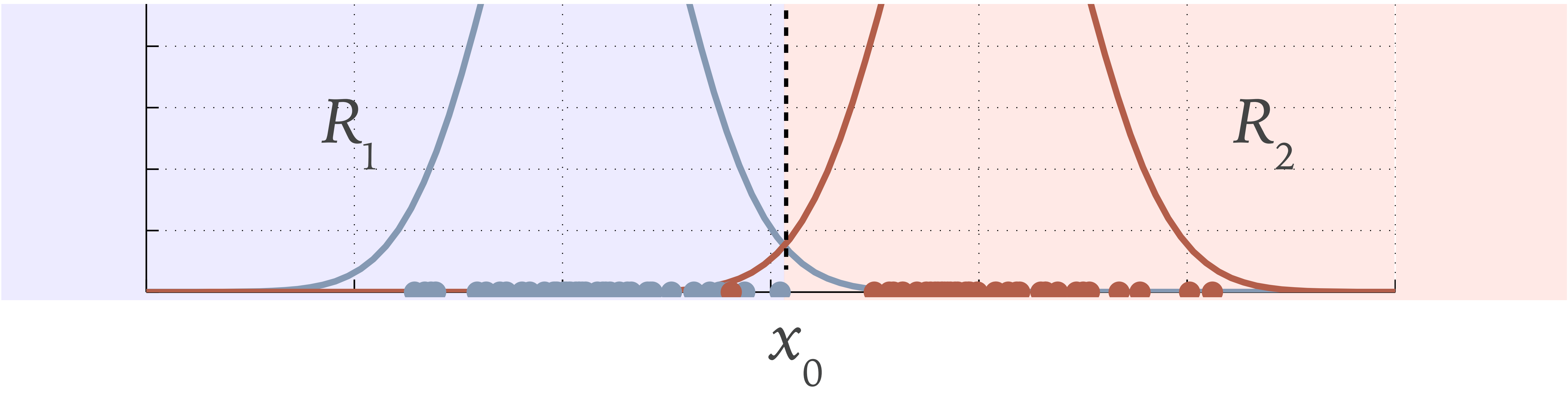
Visualizing the decision

- Assume a Gaussian model for the classes
  - Likelihood: $P(x \mid \omega_i) = \mathcal{N}(x \mid \mu_i, \sigma_i)$



Errors in classification

- We can't win all the time though
  - Some inputs will be misclassified

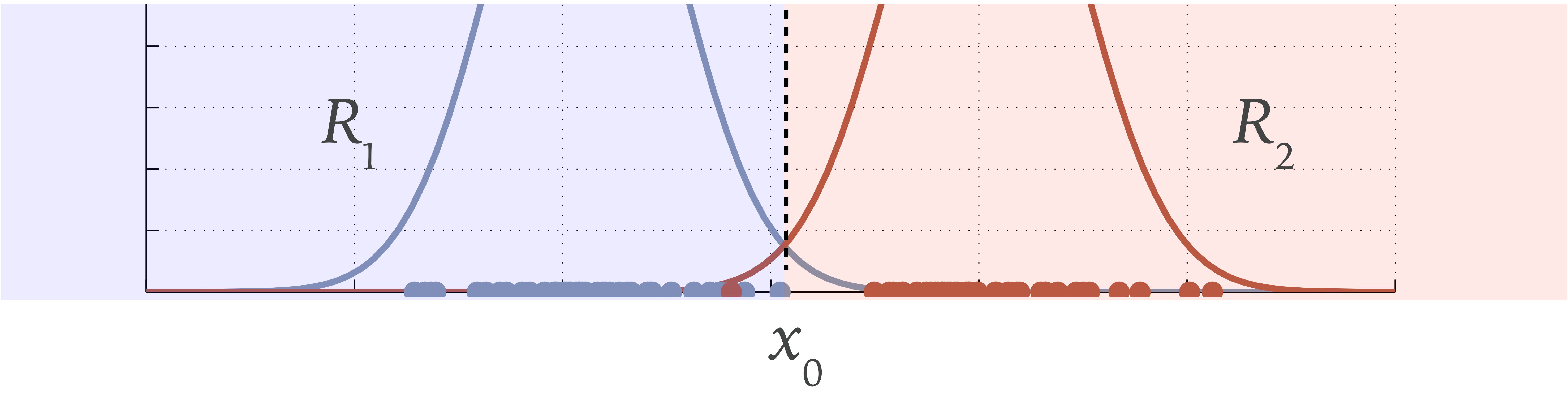


- What are the errors?

$$\varepsilon_2 = \int_{-\infty}^{x_0} P(x \mid \omega_2) P(\omega_2)\, dx, \quad \varepsilon_1 = \int_{x_0}^{\infty} P(x \mid \omega_1) P(\omega_1)\, dx$$

Minimizing misclassifications

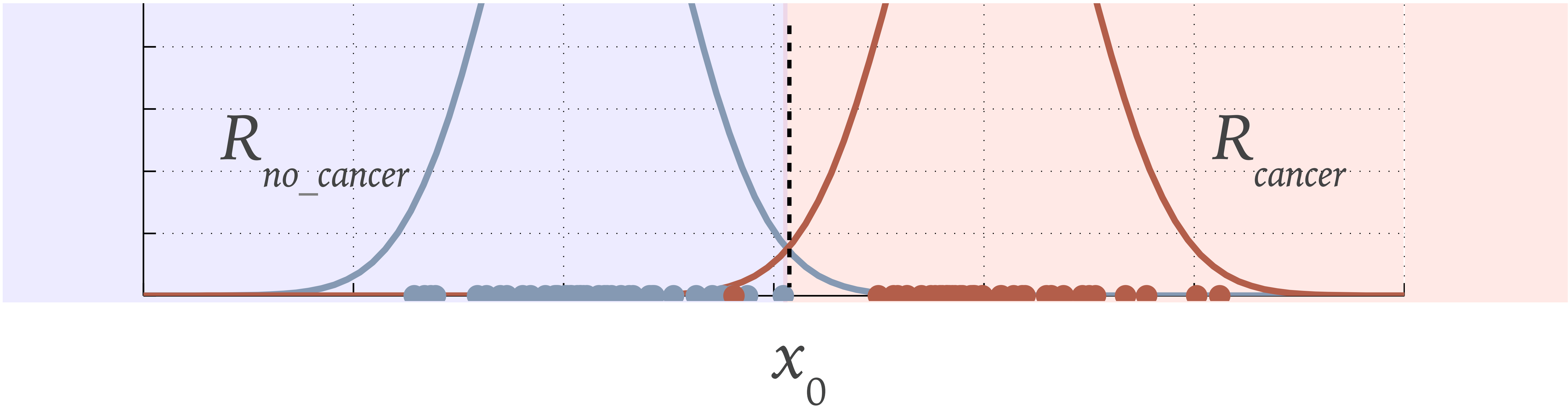
- The Bayes classification rule minimizes these potential misclassifications



- Can you do any better by moving the line?

Minimizing risk

- Not all errors are equal!
  - e.g. medical diagnoses



- Misclassification can be tolerable, or not, depending on the assumed risks

Adding a cost term

- Implement a "loss" factor to each decision to compute "risk" for each class

$$r_j = \sum_i \lambda_{ji} \int_{R_i} P(x \mid \omega_j) dx$$

  - The λ's specify how costly each decision is
    - λ_{ij} is cost for samples in region i while being of class j
- Choose regions to minimize overall risk

$$r = \sum_j r_j P(\omega_j) = \sum_i \int_{R_i} \sum_j \lambda_{ji} P(x \mid \omega_j) P(\omega_j) dx$$

New decision process

- Assign x to ω_1 if

$$(\lambda_{21}-\lambda_{22})P(x\,|\,\omega_2)P(\omega_2)<(\lambda_{12}-\lambda_{11})P(x\,|\,\omega_1)P(\omega_1)$$

Boost ω_2 posterior

Boost ω_1 posterior

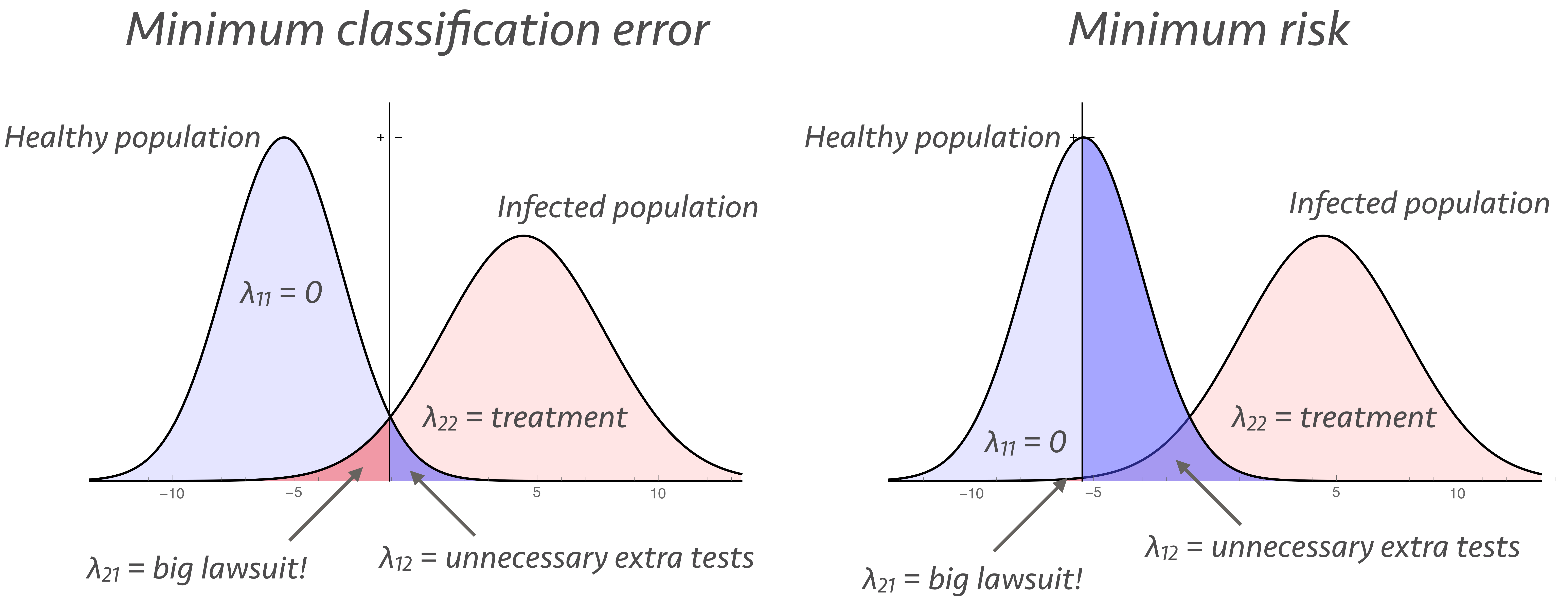
  - and vice-versa
- Or using the *likelihood ratio test*:

$$x\in\omega_1 \quad \text{if} \quad \frac{P(x\,|\,\omega_1)}{P(x\,|\,\omega_2)}>\frac{P(\omega_2)\left(\lambda_{21}-\lambda_{22}\right)}{P(\omega_1)\left(\lambda_{12}-\lambda_{11}\right)}$$

  - and vice-versa

Example case

- A deadly disease detector
  - Optimizing the cost of each decision; note that $\lambda_{21} \gg \lambda_{12}$



True/False – Positives/Negatives

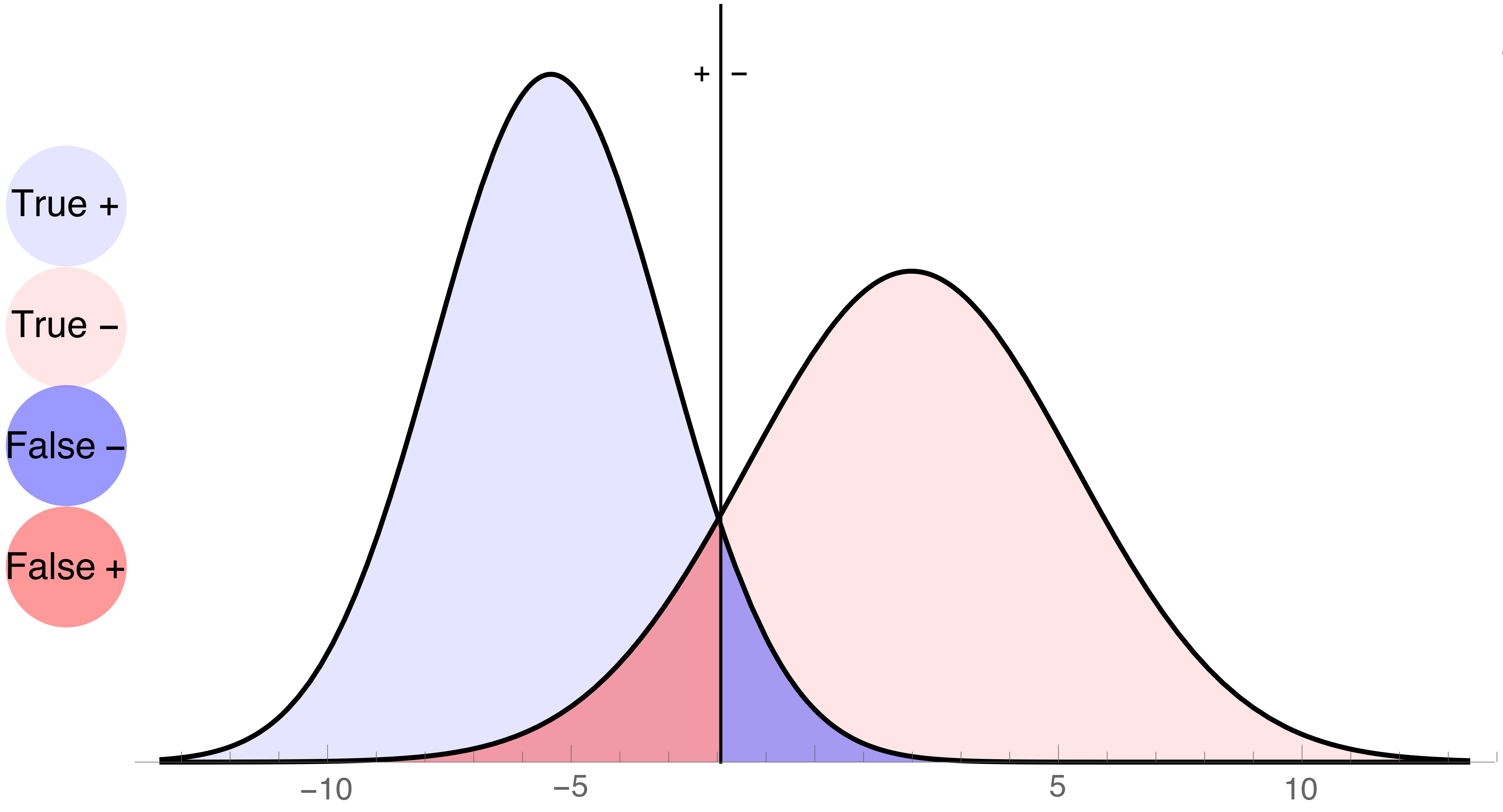
- Naming the outcomes

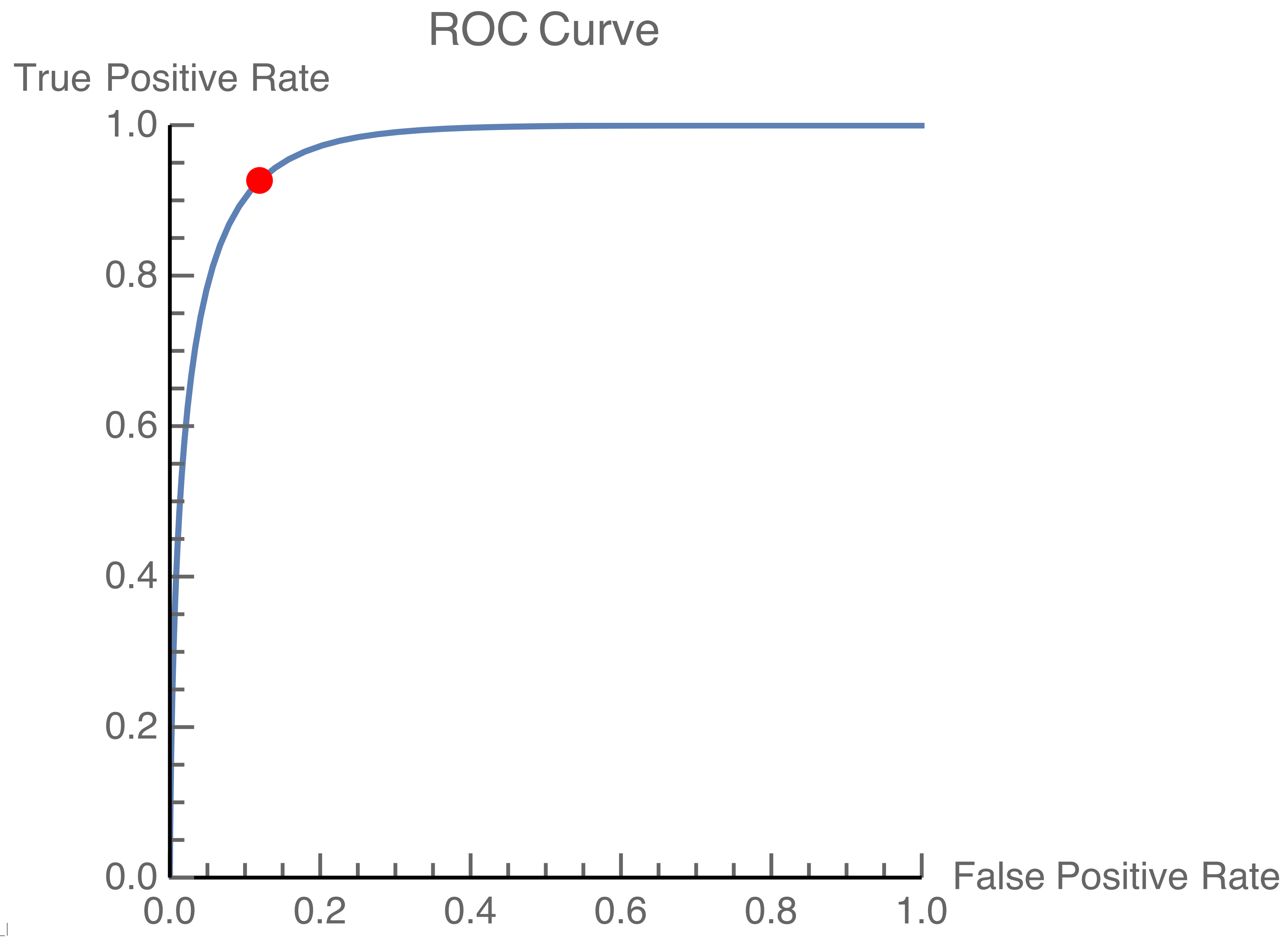
Classifying for ω_1	x is ω_1	x is ω_2
x classified as ω_1	**True positive**	**False positive**
x classified as ω_2	**False negative**	**True negative**

- False positive / False alarm / Type I error
- False negative / Miss / Type II error

Receiver Operating Characteristic

- Visualizing how well we can expect to do





Classifying Gaussian data

- Remember that we need the class likelihood to make a decision
  - For now let's assume that:

$$P(x \mid \omega_i) = \mathcal{N}(x \mid \mu_i, \sigma_i)$$

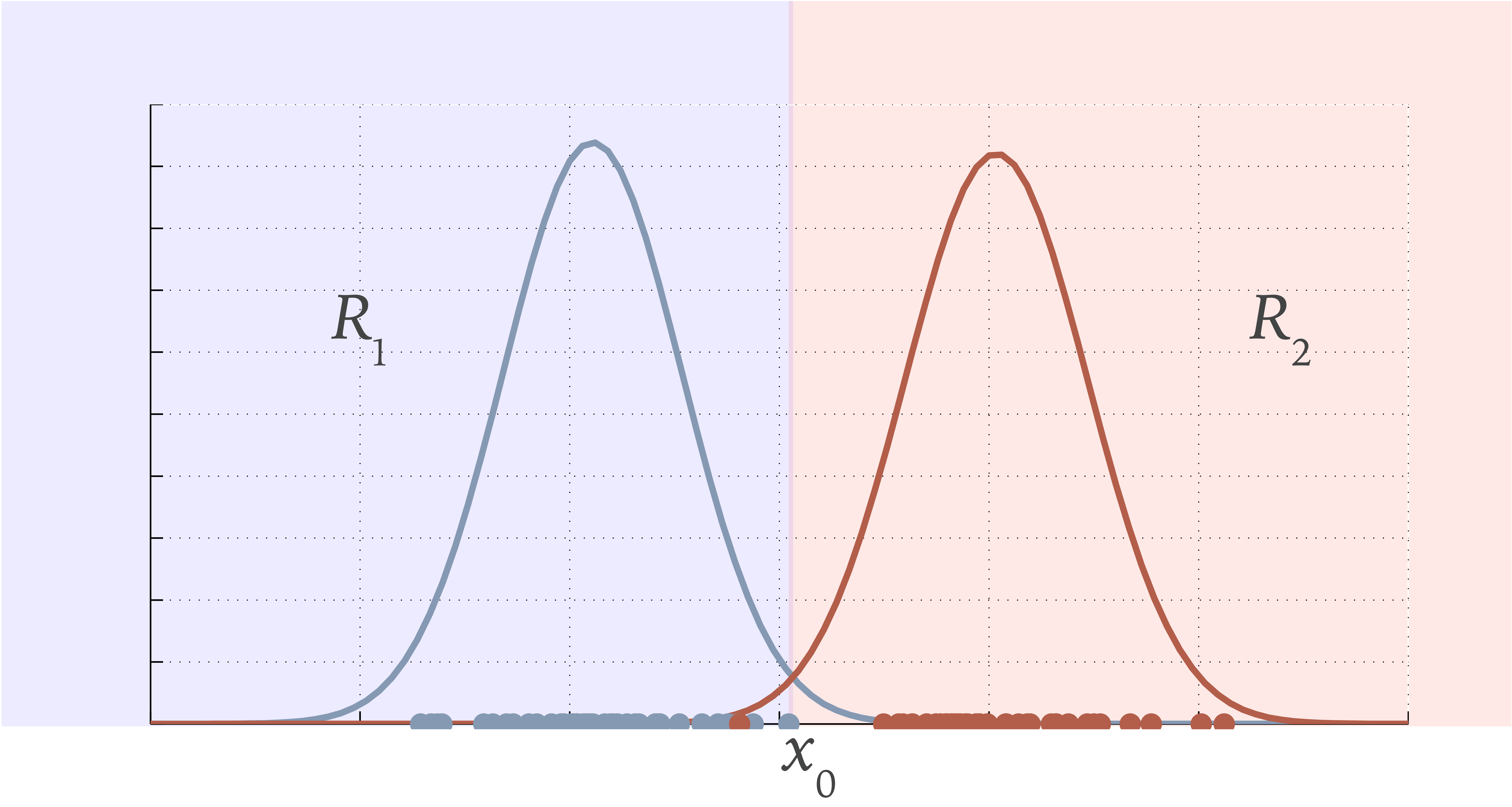
  - i.e. that the input data is Gaussian distributed

Overall methodology

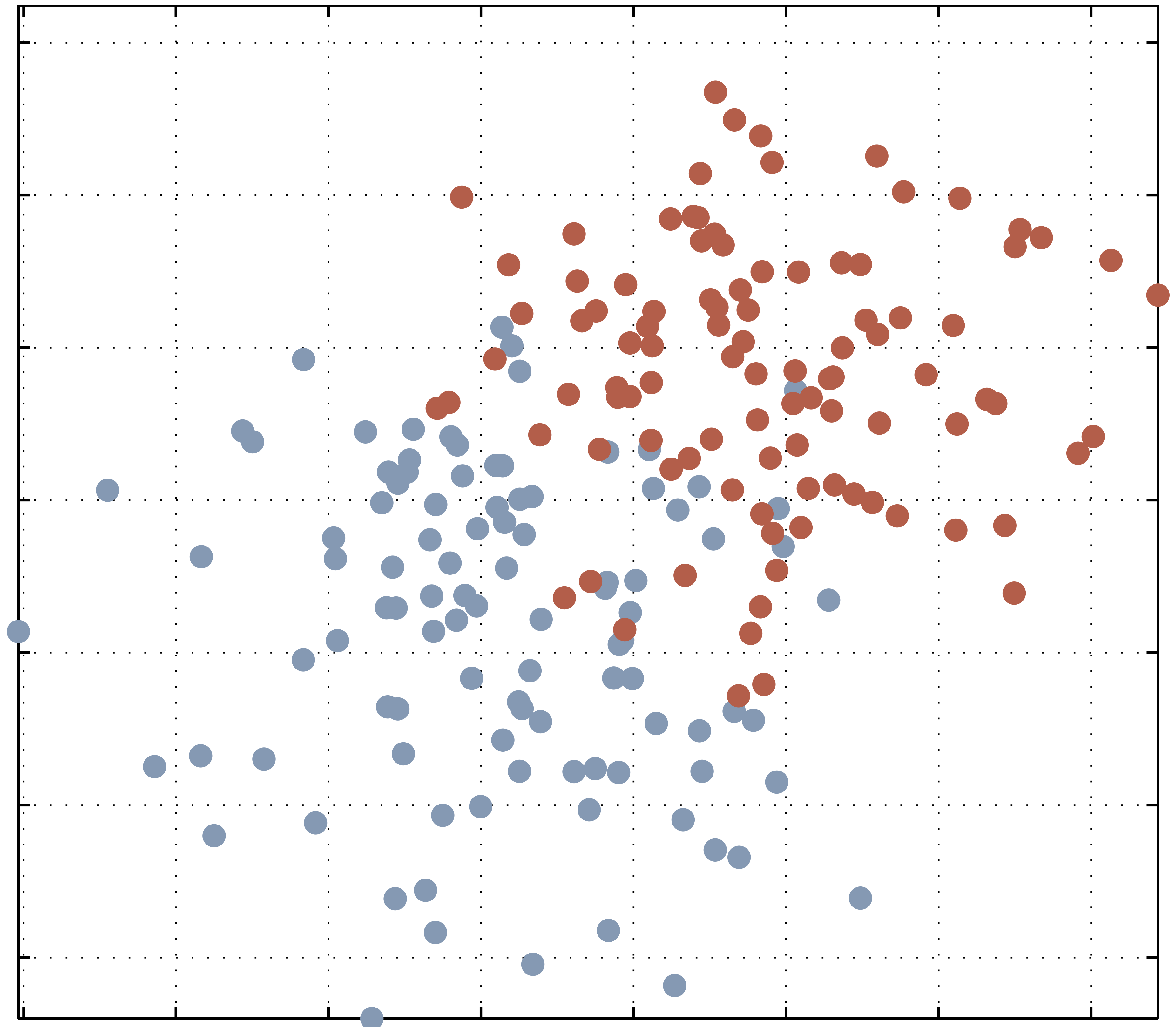
- Obtain training data
- Fit a Gaussian model to each class
  - Perform parameter estimation for mean, variance and class priors
- Define decision regions based on models and any given constraints

1D example

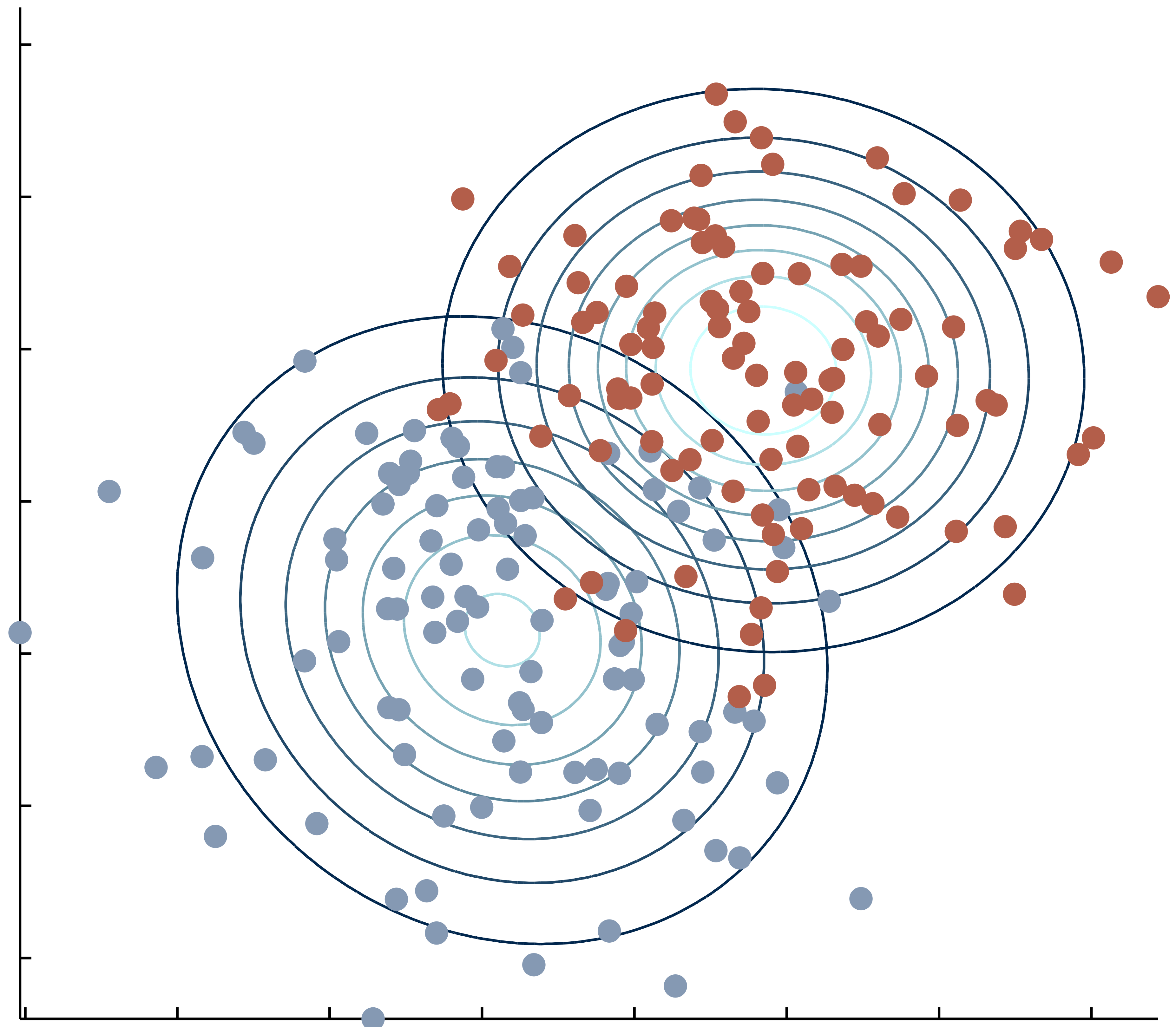
- The decision boundary will always be a point separating the two class regions



2D example



2D example fitted Gaussians



Gaussian decision boundaries

- The decision boundary is defined as:

$$P(\mathbf{x} \mid \omega_1)P(\omega_1) = P(\mathbf{x} \mid \omega_2)P(\omega_2)$$

- Replace likelihoods with Gaussians and solve to find what the boundary looks like

Discriminant functions

- Define a set of functions $g_i(\mathbf{x})$ so that:

$$\text{classify } \mathbf{x} \text{ in } \omega_i \text{ if } g_i(\mathbf{x}) > g_j(\mathbf{x}), \forall i \neq j$$

- Decision boundaries are now defined as:

$$g_{ij}(\mathbf{x}) \equiv \left(g_i(\mathbf{x}) = g_j(\mathbf{x})\right)$$

Discriminant functions for Gaussians

- We remove the exponentiation:

$$g_i(\mathbf{x}) = \log(P(\mathbf{x}\,|\,\omega_i)P(\omega_i)) = \log P(\mathbf{x}\,|\,\omega_i) + \log P(\omega_i)$$
$$= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^\top \cdot \Sigma_i^{-1} \cdot (\mathbf{x}-\boldsymbol{\mu}_i) + \log P(\omega_i) + C_i$$
$$= \frac{1}{2}\left[-\mathbf{x}^\top \cdot \Sigma_i^{-1} \cdot \mathbf{x} + \mathbf{x}^\top \cdot \Sigma_i^{-1} \cdot \boldsymbol{\mu}_i - \boldsymbol{\mu}_i^\top \cdot \Sigma_i^{-1} \cdot \boldsymbol{\mu}_i + \boldsymbol{\mu}_i^\top \cdot \Sigma_i^{-1} \cdot \mathbf{x}\right]$$
$$+ \log P(\omega_i) + C_i$$

- The decision boundaries $g_i(\mathbf{x}) = g_j(\mathbf{x})$ will be *quadrics*

Back to the data

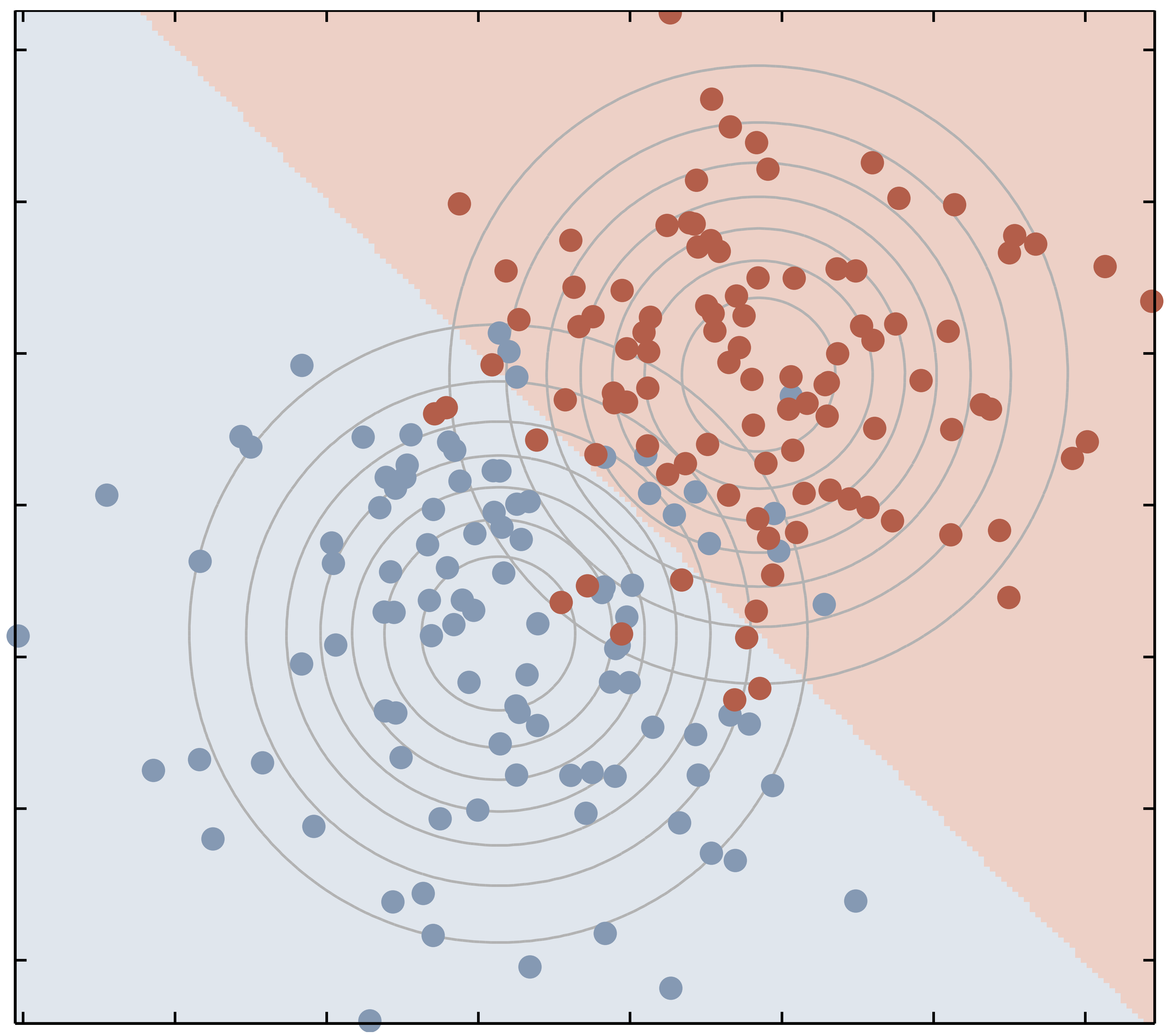
- $\Sigma_i = \sigma^2 \mathbf{I}$ produces line boundaries

- Discriminant:

$$g_i(\mathbf{x}) = \mathbf{w}_i^\top \cdot \mathbf{x} + b$$

$$\mathbf{w}_i = \boldsymbol{\mu}_i / \sigma^2$$

$$b = -\frac{\boldsymbol{\mu}_i^\top \cdot \boldsymbol{\mu}_i}{2\sigma^2} + \log P(\omega_i)$$



Quadratic boundaries

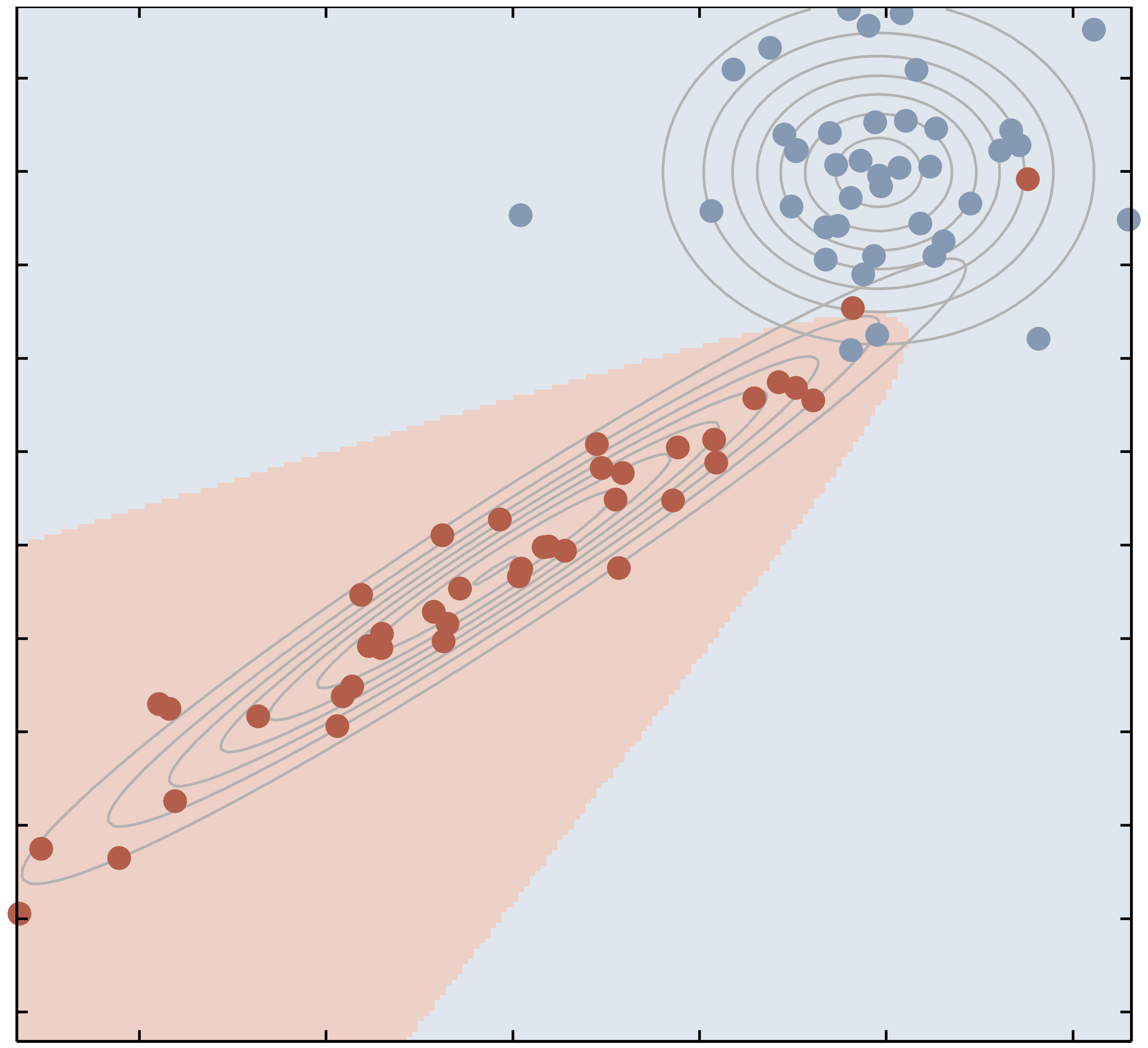
- Arbitrary covariance matrices can produce more elaborate boundaries

$$g_i(\mathbf{x}) = \mathbf{x}^\top \cdot \mathbf{W}_i \cdot \mathbf{x} + \mathbf{w}_i^\top \cdot \mathbf{x} + w_i$$

$$\mathbf{W}_i = -\frac{1}{2}\mathbf{\Sigma}_i^{-1}$$

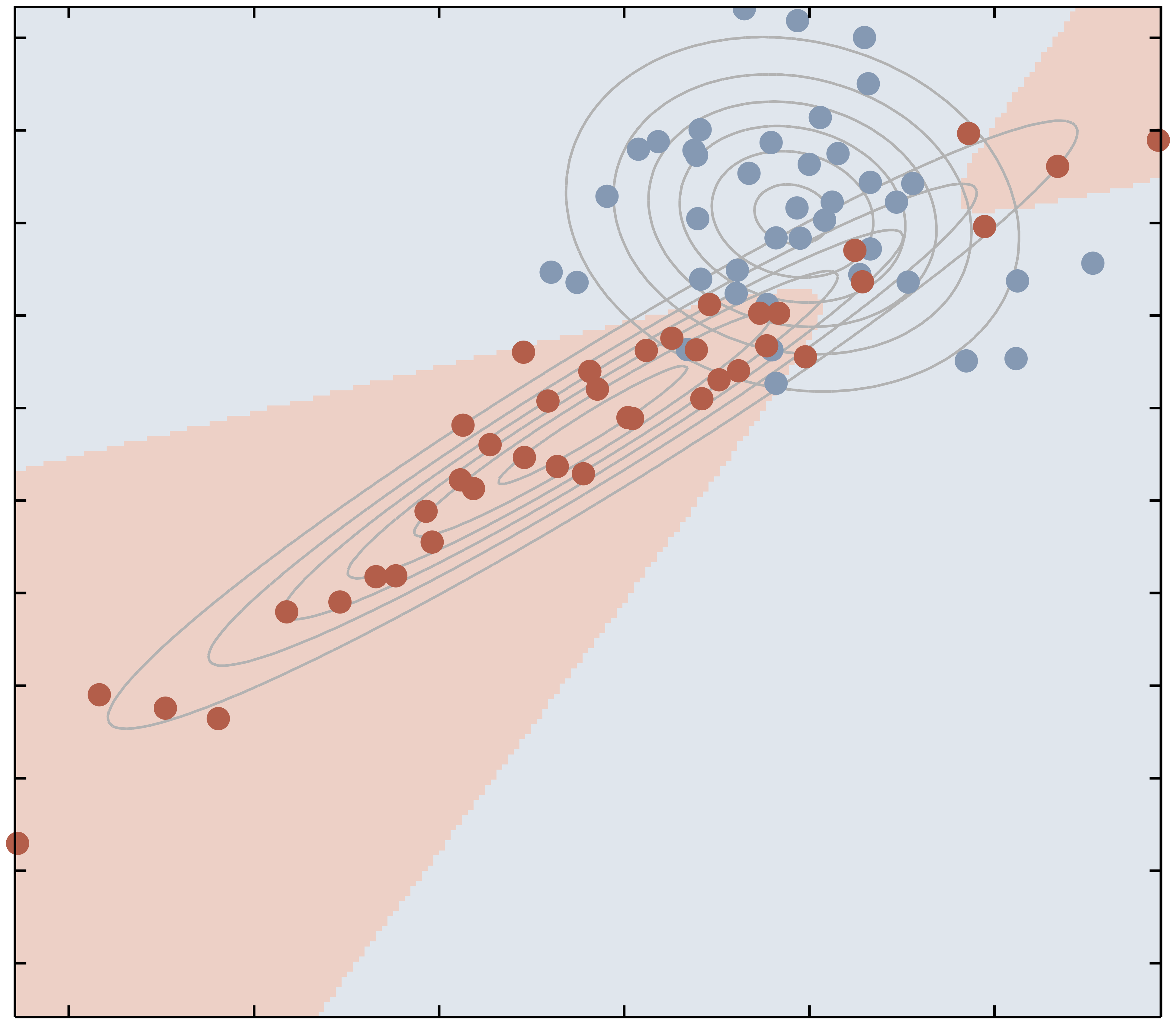
$$\mathbf{w}_i = \mathbf{\Sigma}_i^{-1} \cdot \boldsymbol{\mu}_i$$

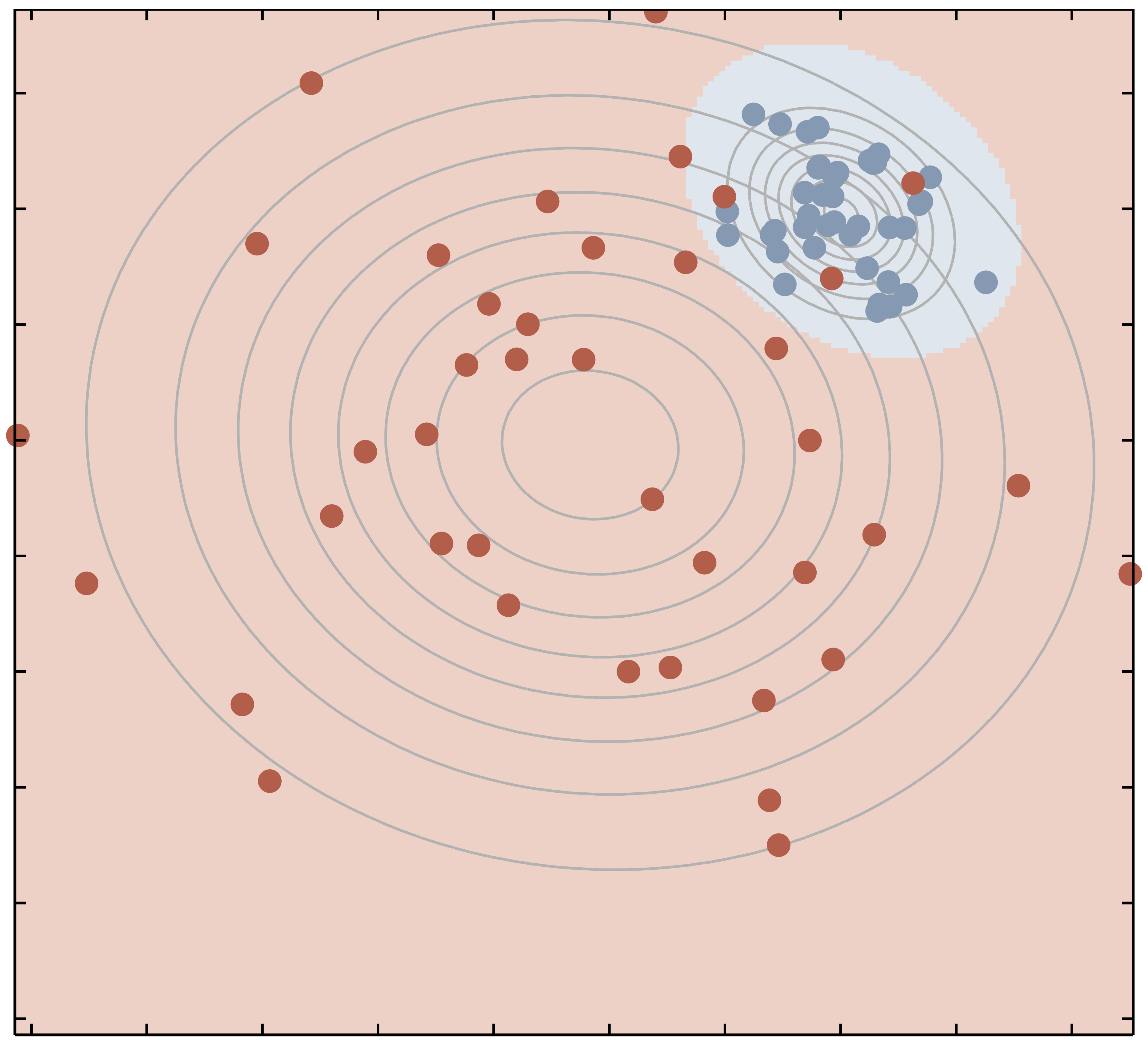
$$w = -\frac{1}{2}\boldsymbol{\mu}_i^\top \cdot \mathbf{\Sigma}_i^{-1} \cdot \boldsymbol{\mu}_i - \frac{1}{2}\log\left|\mathbf{\Sigma}_i\right| + \log P(\omega_i)$$



Quadratic boundaries

- Arbitrary covariance matrices can produce more elaborate boundaries





Naïve Bayes classifier

- Dimensionality issues
  - For large dimensions the Gaussian estimate will require a lot of data! Order N dimensions
- Naïve Bayes classifier assumes independence across dimensions

Naïve Bayes classifier

- Each dimension is sampled independently
  - Thus we don't require many training samples

$$P(\mathbf{x} \mid \omega_i) = \prod_j P(x_j \mid \omega_i)$$

- Overall classification is:

$$\omega = \arg\max_{\omega_i} \prod_j P(x_j \mid \omega_i)$$

  - Not elegant, but reasonably reliable
    - Looks familiar?

A different perspective

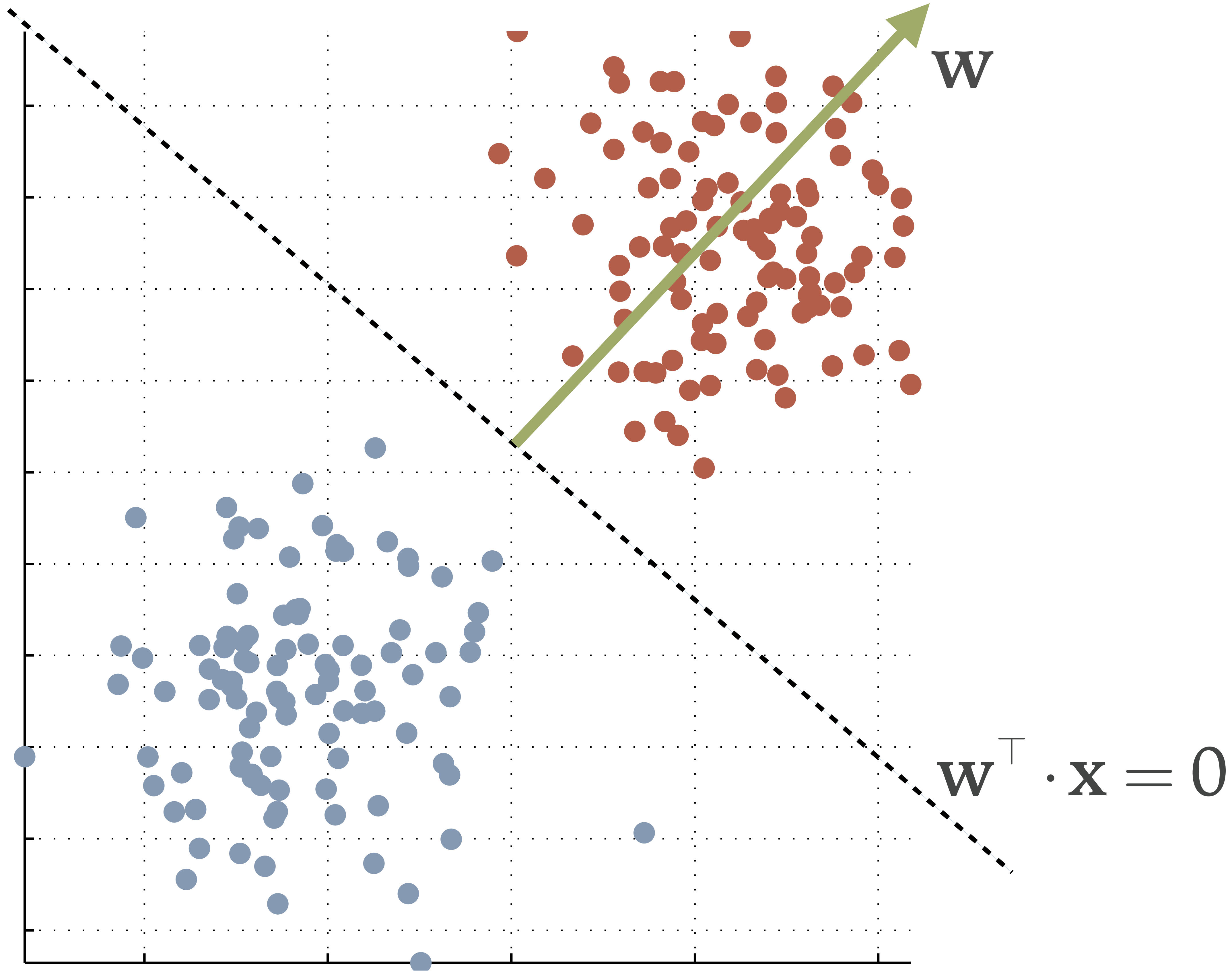
- Obtaining the discriminant function directly
  - Training data $\mathbf{x}_i$, labels $y_i \in \{-1,1\}$
  - Linear discriminant $\mathbf{w}$ and bias b

$$y_i = \mathbf{w}^\top \cdot \mathbf{x}_i + b$$

  - Or we can skip the b and set $\mathbf{x} = [\mathbf{x} \,;\, 1]$ in which case $\mathbf{w} = [\mathbf{w}, b]$
- This is the same as the discriminant function for isotropic Gaussians

Linear classifiers

- Directly defines the class boundary line



Approach 1: The Perceptron

- Assume there is a solution
- Then find $\mathbf{w}$ such that:

$$\mathbf{w}^\top \cdot \mathbf{x}_i > 0 \;\; \text{if} \;\; \mathbf{x}_i \in \omega_1$$
$$\mathbf{w}^\top \cdot \mathbf{x}_i < 0 \;\; \text{if} \;\; \mathbf{x}_i \in \omega_2$$

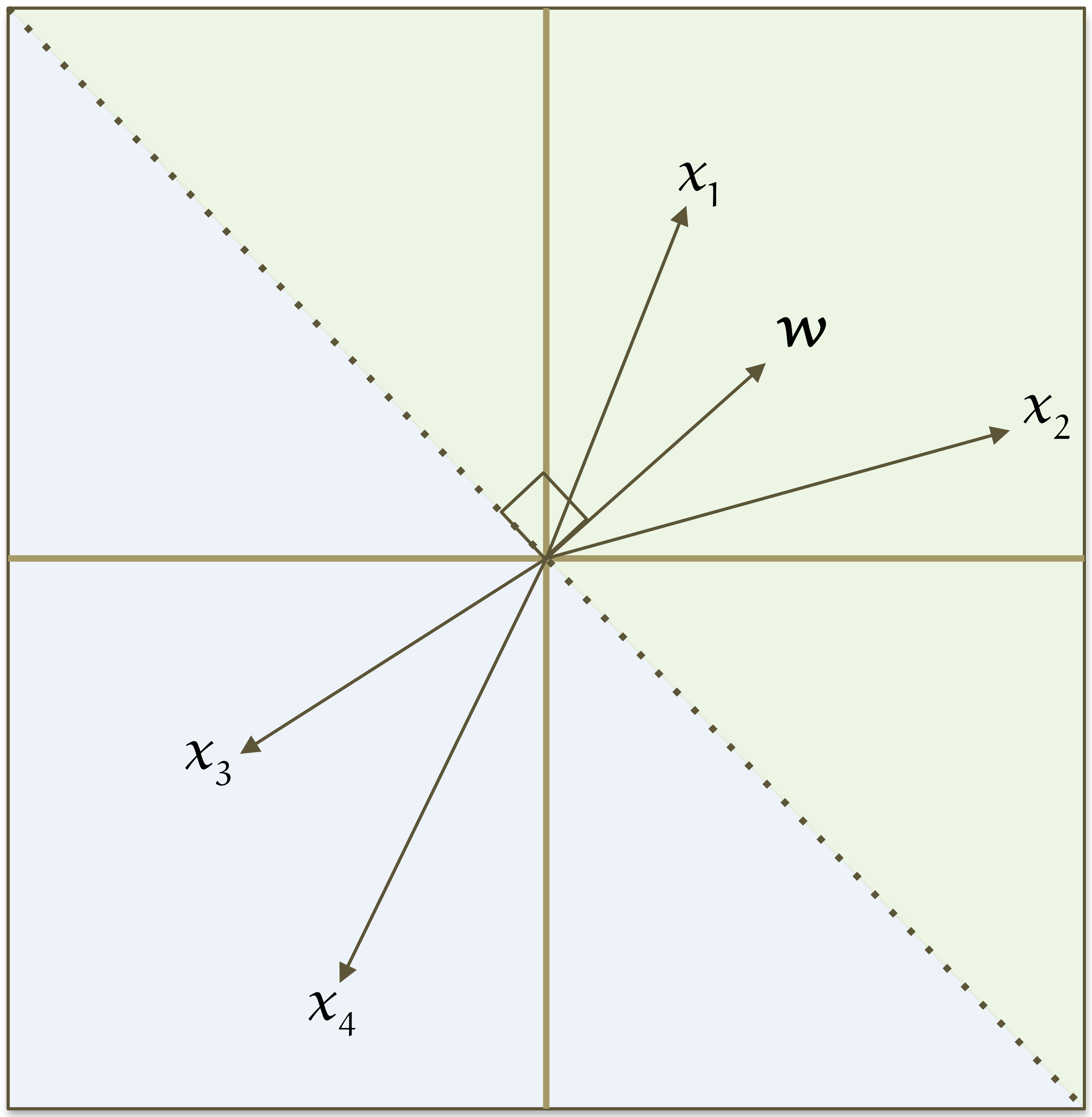
- How do we solve this?

A simple update algorithm

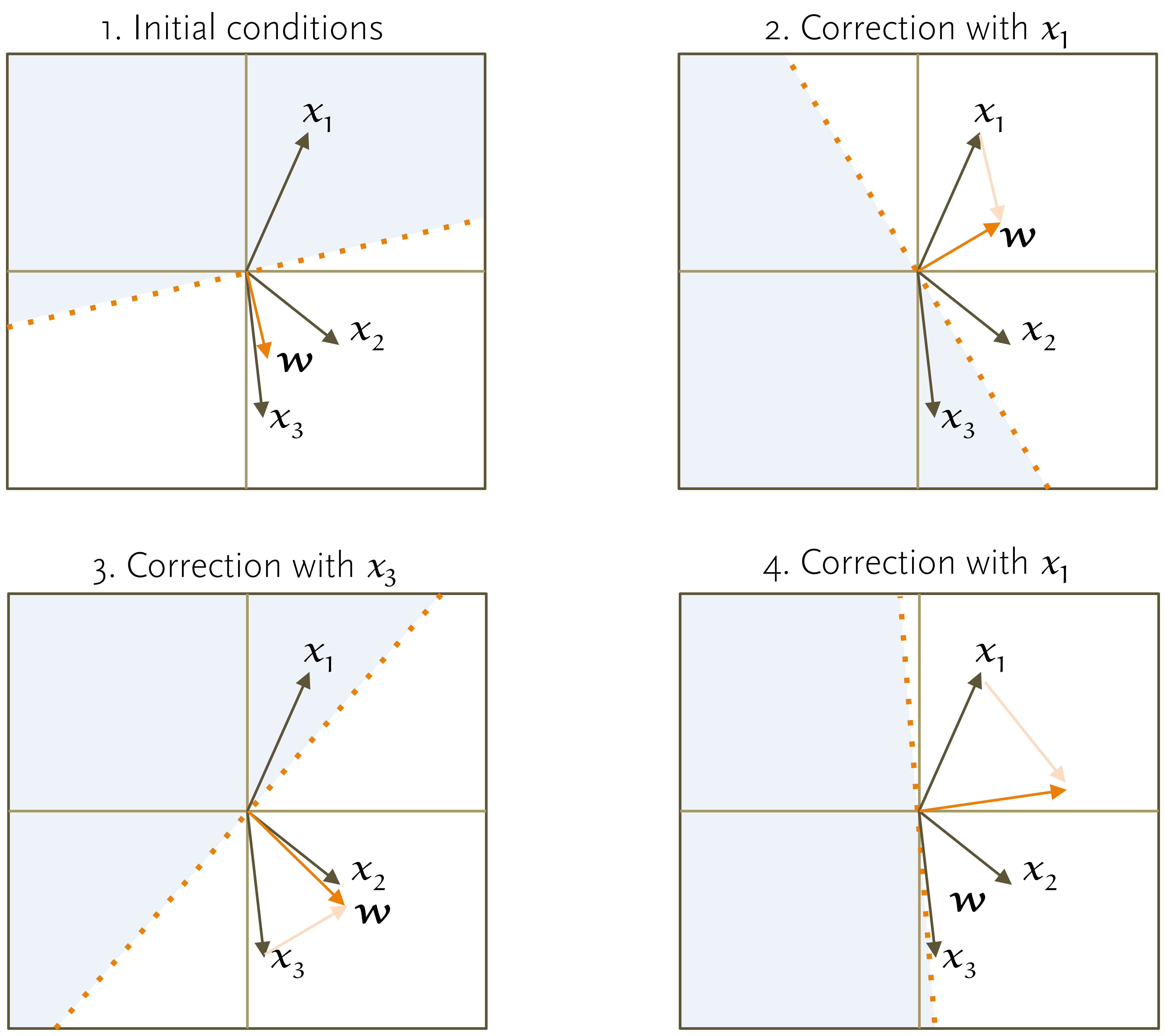
- Using corrections
- For all vectors $\mathbf{x}$:
  - If $\operatorname{sgn}\left(\mathbf{w}^\top \cdot \mathbf{x}_i\right) = y_i$
    - Do nothing
  - If $\operatorname{sgn}\left(\mathbf{w}^\top \cdot \mathbf{x}_i\right) = -y_i$
    - Then $\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i, \quad 0 < \eta < 1$
  - Repeat until no error (or progress) is made

What does this mean?

- $\mathbf{w}$ is normal to the boundary line
- To produce $\mathbf{w}^\top \cdot \mathbf{x}_i > 0$
  - Has to be within 90° of positive data
- To produce $\mathbf{w}^\top \cdot \mathbf{x}_i < 0$
  - Has to be outside of 90° for negative data



Looking at one class



Approach 2: Using a cost function

- Minimize the cost function:

$$J(\mathbf{w}) = \sum_{\forall\, \mathrm{sgn}(\mathbf{w}^\top \cdot \mathbf{x}_i) \neq y_i} \delta_i \mathbf{w}^\top \cdot \mathbf{x}_i$$

$$\delta_i = -y_i = \begin{cases} -1, & \text{if } \mathbf{x}_i \in \omega_1 \\ +1, & \text{if } \mathbf{x}_i \in \omega_2 \end{cases}$$

Contribution to cost function

	$\mathbf{x}_i \in \omega_1$ $\delta_i = -1$	$\mathbf{x}_i \in \omega_2$ $\delta_i = +1$
$\mathbf{w}^\top \cdot \mathbf{x}_i > 0$	0	$\delta_i\, \mathbf{w}^\top \cdot \mathbf{x}_i > 0$
$\mathbf{w}^\top \cdot \mathbf{x}_i < 0$	$\delta_i\, \mathbf{w}^\top \cdot \mathbf{x}_i > 0$	0

- Cost function will be zero if all classifications are correct

Gradient descent approach

- Use a gradient descend algorithm:

$$\mathbf{w} = \mathbf{w} - \eta \sum_{\forall \, \mathrm{sgn}(\mathbf{w}^\top \cdot \mathbf{x}_i) \neq y_i} \delta_i \mathbf{x}_i$$

  - Same as the perceptron!
- Proven convergence, fast and small!
  - Many variants exist
  - A core idea behind neural nets

Approach 3: Minimize Squared Error

- We can also directly solve the problem
  - Assign all samples in a matrix $\mathbf{X}$
  - Assign all class labels in vector $\mathbf{y}$
- Solve for $\mathbf{w}$ such that:

$$\mathbf{y} = \mathbf{w}^\top \cdot \mathbf{X}$$

MSE classifier

- Solving for $\mathbf{w}$ we get:

$$\mathbf{w} = \mathbf{y} \cdot \mathbf{X}^{+}$$

- We only need the pseudoinverse of $\mathbf{X}$
  - Robust closed form solution
- This is essentially a regression problem
  - Least-squares solution

Linear classifiers

- As long as we use a Euclidean distance metric, there is a Gaussian assumption

- Linear classifiers are easier to understand
  - But remember what they actually imply!

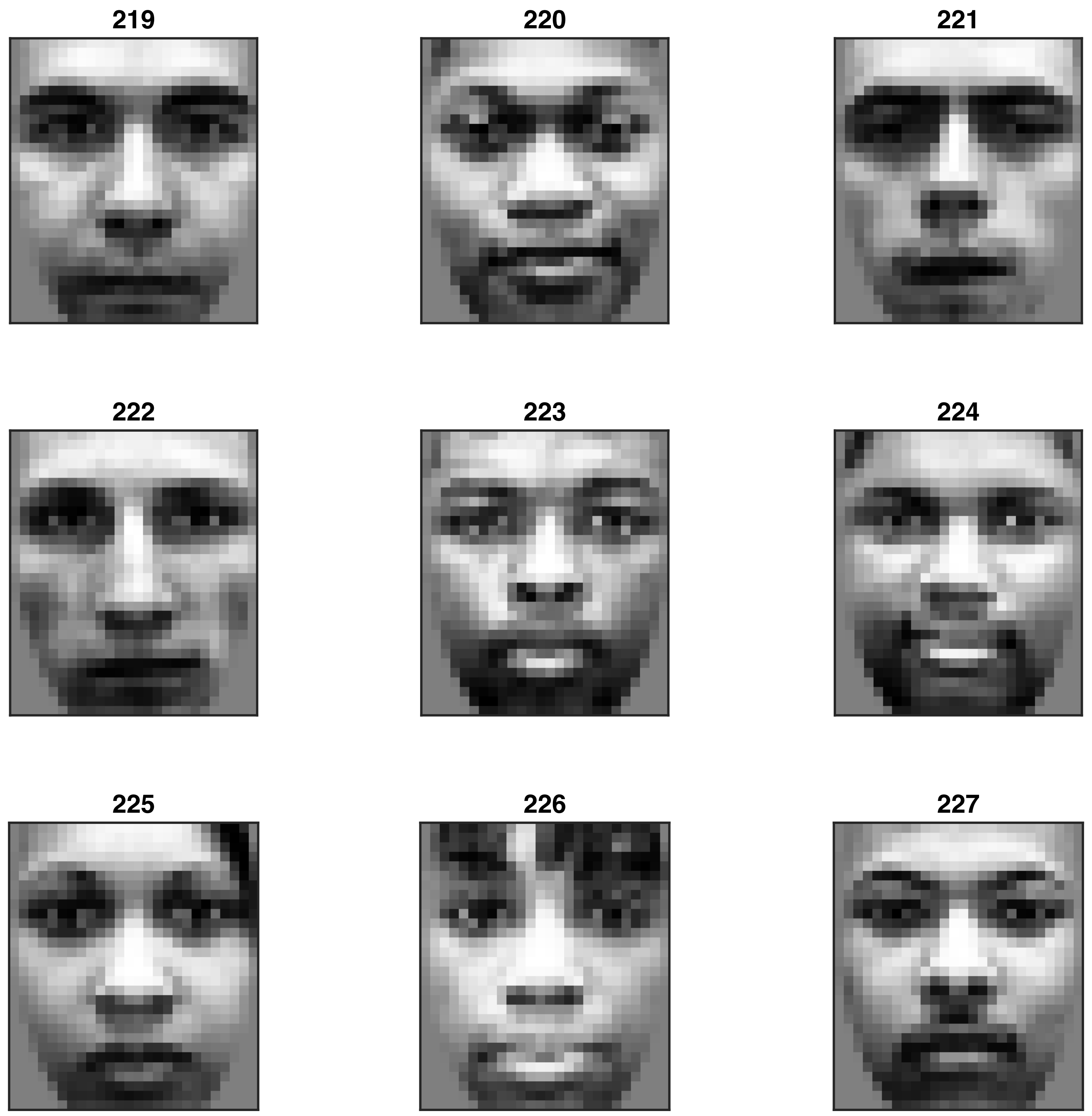
Looking back on detection

- We can now make the detection more elegant
  - No need to look at correlation peaks
- Learn class models from examples
  - Doesn't have to be a single template
- Translate the dot products to likelihoods
  - Apply decision theory to classify in either of the two classes

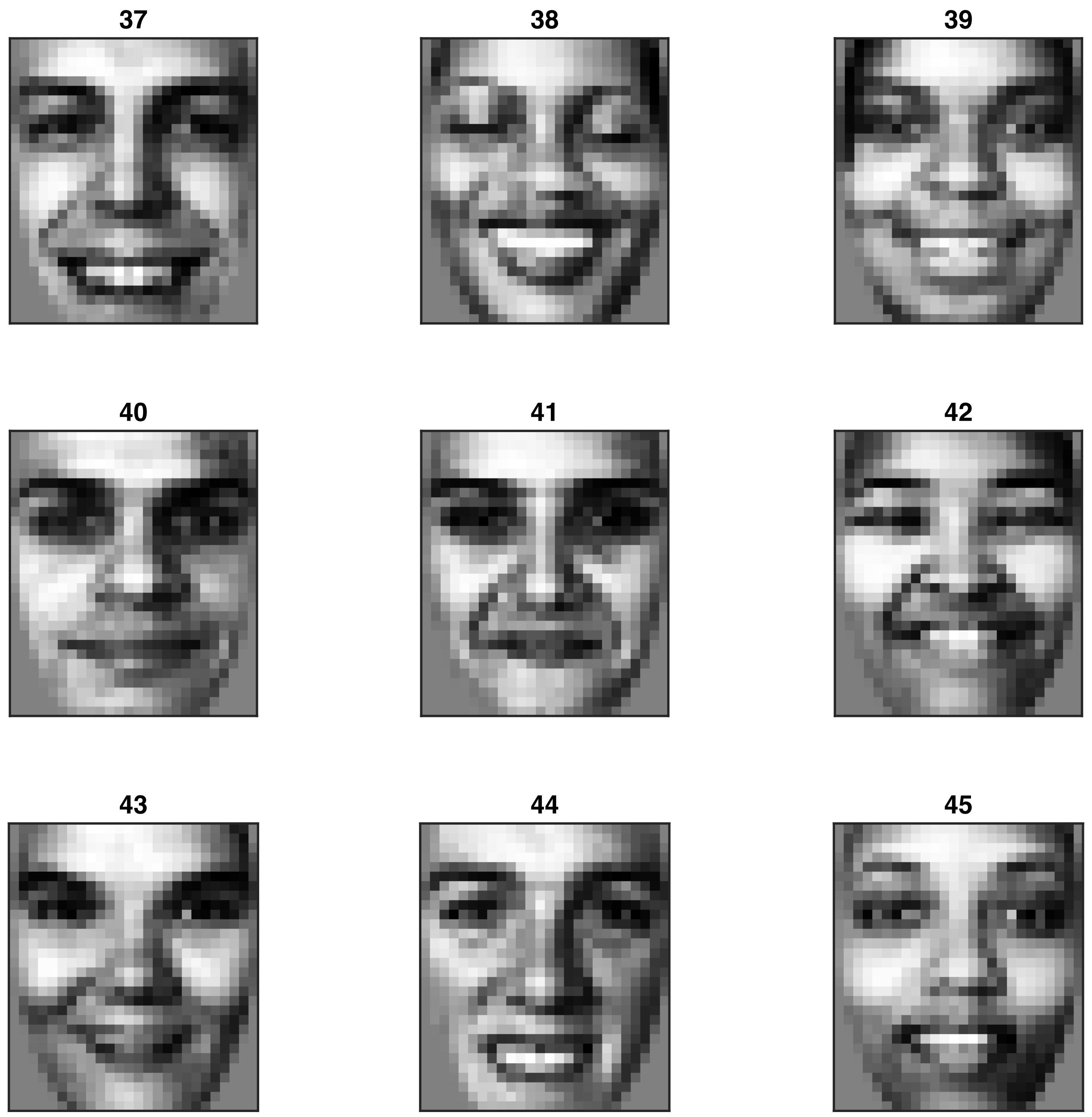
Example: Big smile detector

- Make a classifier that find smiling faces

Some non-smiling samples

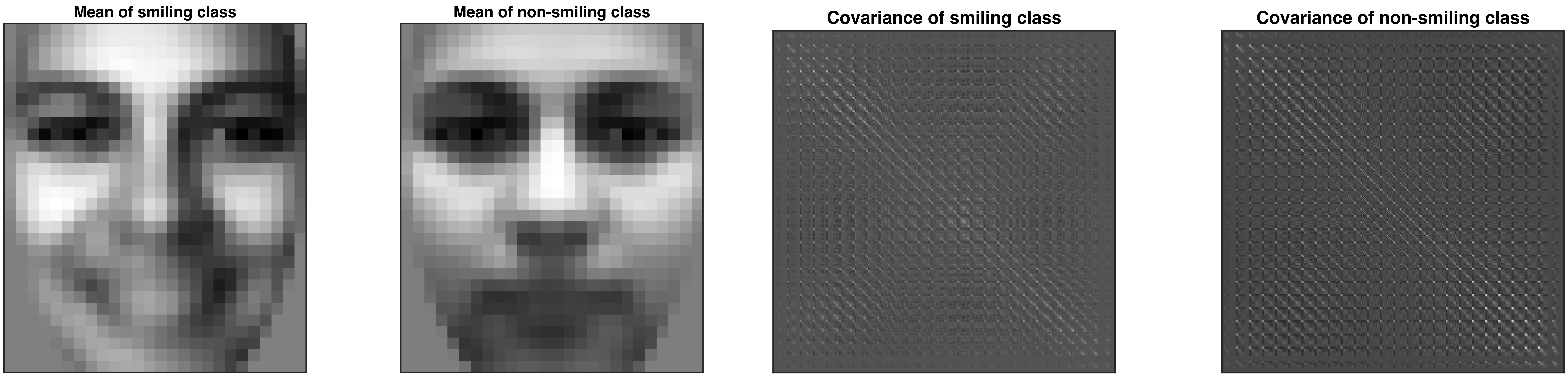


Some smiling samples



Simple way

- Assume classes are Gaussian distributed (are they?)
  - Get means and covariances for each class



Standard ways to classify

- Assume a linear classifier (isotropic Gaussian):

$$P(\mathbf{x}|\omega_i) = \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}_i, \mathbf{I}\right) \propto e^{-(\mathbf{x}-\boldsymbol{\mu}_i)^\top \cdot (\mathbf{x}-\boldsymbol{\mu}_i)}$$

  - Assume equal priors and assign data to maximum likelihood class
    - Get's us okay results

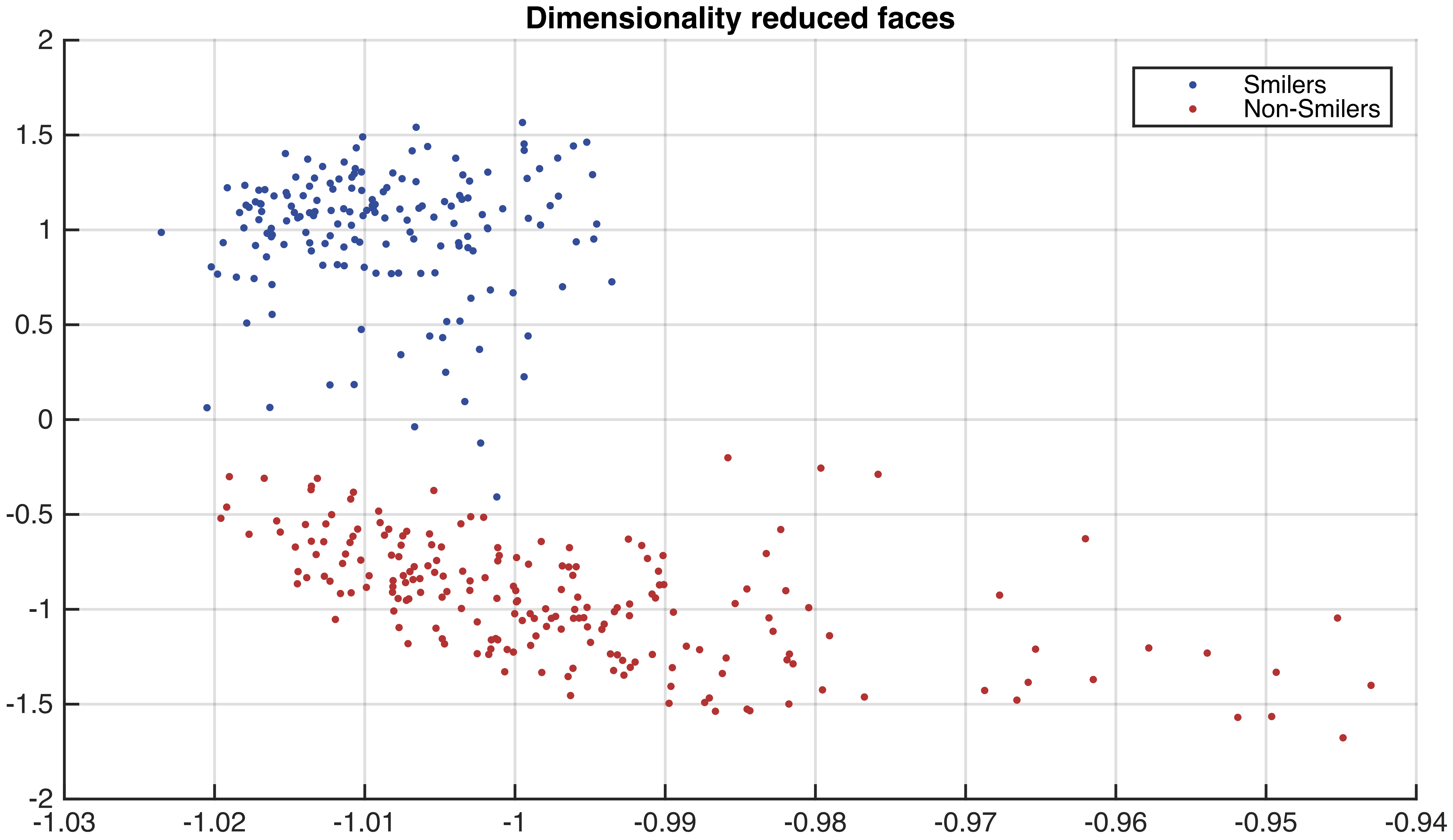
- Assume a quadratic classifier (full or diagonal covariance)

$$P(\mathbf{x}|\omega_i) = \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) \propto e^{-(\mathbf{x}-\boldsymbol{\mu}_i)^\top \cdot \boldsymbol{\Sigma}_i^{-1} \cdot (\mathbf{x}-\boldsymbol{\mu}_i)}$$

  - Oops, the covariance might be non-invertible

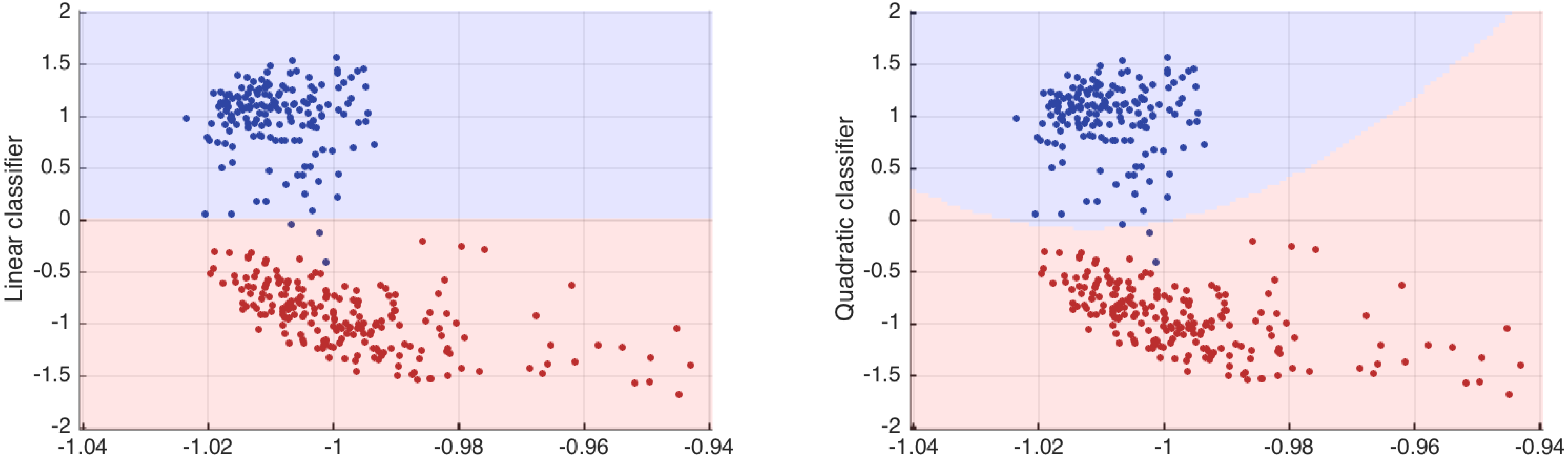
Or we can use features

- Perform PCA on the faces and drop to low dimensions
  - Now the covariance estimate is better behaved



Redo the classification on the low dims

- Faster and easier, and just as good
  - Not crucial in this case, but the quadratic classifier is a bit better



- How can we improve this? Is it worth it?

Recap

- The Bayesian view
  - Risk, decision regions
  - Gaussian classifiers, Naïve Bayes
- Linear classifiers
  - The perceptron, separating hyperplanes

Next lecture

- Support Vector Machines
- Non-linear classifiers
  - Neural nets and Kernels

Reading

- Textbook chapters 2-2.4, 2.5.7 and 3-3.6

Problem set 2

- It's out, in case you missed it …